Kednos PL/I for OpenVMS Alpha™

Release Notes for Version 4.4B and Runtime Library Version 4.4B

This manual provides release information on Kednos PL/I Version 4.4B for OpenVMS Alpha, a software language and the associated Runtime Library Version 4.4B.

Revision/Update Information:	This is a new manual.
Operating System and Version:	For Kednos PL/I for OpenVMS Alpha: OpenVMS Alpha Version 6.2 or higher
Software Version:	Kednos PL/I Version 4.4 for OpenVMS Alpha
Published by:	Kednos Corporation, Pebble Beach, CA, www.Kednos.com

May 2004

Kednos Corporation, makes no representations that the use of its products in the manner described in this publication will not infringe on existing or future patent rights, nor do the descriptions contained in this publication imply the granting of licenses to make, use, or sell equipment or software in accordance with the description.

Possession, use, or copying of the software described in this publication is authorized only pursuant to a valid written license from Kednos Corporation or an anthorized sublicensor.

No responsibility is assumed for the use or reliability of software on equipment that is not listed as supported in the Product Description.

Copyright Kednos Corporation, 1980-2004. All rights reserved.

Copyright ©1980-2004

The following are trademarks of Hewlett Packard Company: Alpha, AXP, DEC, DECnet, DECtalk, DECUS, Digital, IVP, MicroVAX, OpenVMS, RMS, RMS-11, RX50, TK50, VAX, VAX Ada, VAX BASIC, VAX BLISS, VMScluster, CDD Repository, VAX COBOL, VAX DATATRIEVE, VAX DIBOL, VAX DOCUMENT, VAX FORTRAN, VAXinfo, VAX MACRO, VAX Pascal, VAX SCAN, VAXset, VAXTPU, and the DIGITAL logo.

Contents

PREFACE			vi
1	OVERVIEW OF CHANGES MADE IN KEDNOS PL/I VERSION 4.4B AND KEDNOS RUNTIME LIBRARY VERSION 4.4B		1
2	OVERVIEW OF CHANGES MADE IN RUNTIME LIBRARY VERSION 4.4		1
3	OVERVIEW OF CHANGES MADE IN VERSION 4.4 AND RUNTIME LIBRARY VERSION 4.3		1
4	OVERVIEW OF CHANGES MADE IN VERSION 4.3 AND RUNTIME LIBRARY VERSION 4.2		2
5	OVERVIEW OF CHANGES MADE IN VERSION 4.2		2
6	OVERVIEW OF CHANGES MADE IN VERSION 4.1		2
7	INSTALLATION INFORMATION		3
	7.1	Kit Information	3
	7.2	Compression of the PL/I System Interface Library, PLI$STARLET.TLB	3
8	DETAILS OF CHANGES MADE IN VERSION 4.1		3
	8.1	Corrected Documentation Errors	5
9	OVERVIEW OF CHANGES MADE IN VERSION 4.0A		5
10	OVERVIEW OF NEW FEATURES IN VERSION 4.0		10
	10.1	Overview of Language Support	10
	10.2	DEBUG Support	10
	10.3	Differences in VAX PL/I and DEC PL/I Support for Floating-Point Data Types	11
	10.4	Support for Other Language Elements	12

11	OVERVIEW OF CHANGES FROM VAX PL/I		13
	11.1	Corrected Compiler Errors	21
	11.2	Corrected Documentation Errors	22
12	KNOWN PROBLEMS AND RESTRICTIONS WITH VERSION 4.1		23
13	KNOWN PROBLEMS AND RESTRICTIONS WITH VERSION 4.0		23
	13.1	Compiler Known Errors and Methods of Avoidance	23
	13.2	Run-Time Library Known Errors and Methods of Avoidance	27
	13.3	Known Errors in the PL/I for OpenVMS Systems Documentation	27

EXAMPLES

1	Displaying Arguments Passed to a Condition Handler (AXP)	28

Preface

This document contains release information on Kednos PL/I Version 4.4B for OpenVMS Alpha Systems. Release information is necessary for gaining the best results from Kednos PL/I Version 4.4B on the OpenVMS Alpha operating system. Kednos recommends that all users read this information. Kednos Corporation also recommends that all users review the section of this manual entitled "Known Errors and Restrictions" before submitting an SPR.

1 Overview of Changes Made in Kednos PL/I Version 4.4B and Kednos Runtime Library Version 4.4B

Kednos PL/I Version 4.4B and Kednos Runtime Library Version 4.4B contain the following enhancements and fixes:

- A defect has been corrected whereby the signal SS$_IMGDMP was being intercepted and displayed (chained with PLI$_ERROR) rather than passed to VMS for processing.
- A defect has been corrected whereby signals displayed chained with PLI$_ERROR contained extraneous error messages chained onto the end.
- A separate Hobbyist license from Kednos is now required in addition to the OpenVMS Hobbyist license from HP.

2 Overview of Changes Made in Runtime Library Version 4.4

Kednos Runtime Library Version 4.4 contain the following enhancements and fixes:

- A defect has been corrected whereby a program which did output to a disk file (including SYS$OUTPUT in batch) might omit the last line of output.

3 Overview of Changes Made in Version 4.4 and Runtime Library Version 4.3

Kednos PL/I Version 4.4 and Runtime Library Version 4.3 contain the following enhancements and fixes:

- The product name has been changed to Kednos PL/I for OpenVMS Alpha.
- A defect has been corrected whereby compilation of a program with a large amount of static data would fail with the error: %PLIG-F-ASSERTION, Handler context data is allocated at a negative offset
- A defect has been corrected whereby text output of G-floating and T-floating data was restricted to the range of D-floating variables.
- A defect has been corrected whereby compiling the SCREEN.PLI module from the PHONE sample application would fail with the error: %PLIG-F-ASSERTION, GEM_GX_ENUMERATE - initial set empty for
- A defect has been corrected whereby certain programs from the IVP which happened to not have condition handlers at the outermost routine would hang when any condition was signalled.
- A defect has been corrected whereby compiling certain modules /DEBUG would fail with the error: %PLIG-F-BUGCHECK, Compiler bug check during translation.

- After June 2004, a separate Hobbyist license from Kednos will be required, beyond simply having the OpenVMS-Hobbyist license from HP.

4 Overview of Changes Made in Version 4.3 and Runtime Library Version 4.2

DEC PL/I Version 4.3 and Runtime Library Version 4.2 contain the following enhancements and fixes:

- The VMS Hobbyist license is now supported.
- Storage allocations of PICTURE types is corrected.
- Comma delimited fields in GET STRING have been fixed.
- Nested %INCLUDE statements are now correctly supported.
- To move a DEC PL/I program to another system where PL/I is not installed, you will need to install a separate DPLI$RTLSHR kit, which is not delivered as part of OpenVMS.

5 Overview of Changes Made in Version 4.2

DEC PL/I Version 4.2 contains the following enhancements and fixes:

- Performance of operations on BIT(1) data has been enhanced to use in-line operations instead of making (more expensive) calls to OTS routines.
- The PL/I compiler and the PL/I System Interface Library, PLI$STARLET.TLB, have been modified to support use of the /ALIGNED command-line qualifier with this library. Since this corrents a structure layout problem, if you use the /ALIGNED command-line qualifier, you should recompile all sources with the PL/I 4.2 compiler.

6 Overview of Changes Made in Version 4.1

DEC PL/I Version 4.1 contains the following enhancements and fixes:

- VMS Version 7.0 Support

  DEC PL/I Version 4.1 contains a new runtime that runs on VMS Version 6.2 or higher, including VMS Version 7.0. PL/I code compiled on prior versions of the product will run on this version. However, PL/I code must be compiled using DEC PL/I Version 4.1 to run on VMS Version 7.0 or higher.

  The PL/I System Interface Library, PLI$STARLET.TLB, has been rebuilt to run on VMS Version 6.2 or higher, including VMS Version 7.0

- Condition Handler Improvements

Performance of condition handlers has been enhanced. See the Details of Changes Made in Version 4.1 section for details.

- Miscellaneous Fixes

  See the Details of Changes Made in Version 4.1 section for details.

7 Installation Information

This section describes changes to the installation procedure. Please read this section carefully prior to installing DEC PL/I.

7.1 Kit Information

The DPLI041 kit supplies the DEC PL/I Run-Time Library (RTL) to allow correct linking of programs compiled with DEC PL/I. The RTL will be installed if you do not have one or if the copy on your system is older than the copy on this kit.

7.2 Compression of the PL/I System Interface Library, PLI$STARLET.TLB

Beginning with VAX PL/I Version 2.3 for OpenVMS VAX, the system interface library, PLI$STARLET.TLB, has been shipped in compressed library format, which reduces the amount of space required to store the library but increases the amount of time required to access and retrieve a component of the library, such as a system-service declaration. If you use PLI$STARLET.TLB frequently and you have disk space available, you may want to decompress the library during the installation, when you have the opportunity to do so. If you choose to install the compressed version during installation, you can decompress it later, using the following command:

```
$ LIBRARY/DATA=EXPAND/OUTPUT=SYS$COMMON:[SYSLIB] -
$_ SYS$COMMON:[SYSLIB]PLI$STARLET.TLB
```

The decompressed form of the library is slightly over twice the size of the compressed version.

8 Details of Changes Made in Version 4.1

DEC PL/I Version 4.1 contains fixes to problems reported in DEC PL/I Version 4.0A.

- (DECPLI40A-69)

  Performance of ON-conditions was slower on AXP than on VMS.

  Condition handling support has been redesigned to make better use of Alpha architectural features. This results in improved runtime performance.

  This problem has been corrected in the DEC PL/I Compiler.

- (DECPLI40A-72)

The DEC PL/I compiler was failing to properly parse a GET LIST statement when the target variable contained multiple commas, for example:

```
stringvalue = 'XXX,,,,,,,,,';
GET STRING(stringvalue) LIST(cv1,cv2,cv3,cv4,cv5,cv6,cv7);
```

where cv1...cv7 are CHARACTER(n) VARYING.

stringvalue = 'XXX,,,,,,,,,'; GET STRING(stringvalue) LIST(cv1,cv2,cv3,cv4,cv5,cv6,cv7); where cv1...cv7 are CHARACTER(n) VARYING.

This problem has been corrected in the DEC PL/I Compiler.

- (DECPLI40A-77) (DECPLI40A-336)

  Previously, the following symbols of the $NSADEF module were not available in the SYS$LIBRARY:PLI$STARLET.TLB:

  NSA$_EVENT_TYPE

  NSA$_EVENT_SUBTYPE

  PLI$STARLET.TLB was also missing the definitions for the following new OpenVMS V6.2 routines.

  SYS$PERSONA_ASSUME

  SYS$PERSONA_CREATE

  SYS$PERSONA_DELETE

  These problems were fixed by the updates to make PLI$STARLET.TLB compatible with VMS V7.0.

- (DECPLI40A-548)

  With a particular combination of area size and allocation sizes, it was possible for a second ALLOCATE in an area to corrupt the contents of the memory obtained by the first ALLOCATE in that area.

  This problem has been corrected in the DEC PL/I Compiler.

- (DECPLI40A-548) Nonlocal GOTO statements did not unwind properly when compiled with optimizations.

  This problem has been corrected in the DEC PL/I compiler.

- (DECPLI40A-837)

  A conversion problem occurred in a GET EDIT statement with the STRING option when zero was specified in the input string as follows:

```
TEST: PROCEDURE OPTIONS(MAIN);
    DCL FixedVar FIXED;
    GET STRING('       00') EDIT(FixedVar) (P'-------V99');
END TEST;
```

  This problem has been corrected in the DEC PL/I Compiler.

- (DECPLI40A-981)

  Repeatedly opening and closing an indexed file declared as KEYED and opened for INPUT resulted in a memory leak.

This problem has been corrected in the DEC PL/I Runtime.

8.1 Corrected Documentation Errors

The errors in this section have been corrected in the current version of the documentation (DEC PL/I Version 4.1).

- (DECPLI40A-847)

  The reference manual did not mention the alternative of natural alignment when discussing storage allocation.

  This is corrected in chapter 3 of the reference manual. Section 3.4.3 contains the following statement:

  PL/I allocates storage for an aligned bit-string variable on a byte boundary and reserves an integral number of bytes to contain the variable.

  This is true for the default packed alignment. The following has been added to the documentation:

  If natural alignment is in effect, PL/I allocates storage for an aligned bit-string variable on a longword boundary and reserves an integral number of longwords to contain the variable. See the user manual for information on the /ALIGN and /DATA command line qualifiers, which affect what type of alignment is in effect.

- The last paragraph of section 3.2.4 of the reference manual contained values for T_floating data in the range 2.225073859D-380 to 1.797693135D308. The letter D in both of these numbers is invalid, and should be E.

9 Overview of Changes Made in Version 4.0A

DEC PL/I Version 4.0A contains fixes to problems reported in DEC PL/I Version 4.0.

- (3-CFS.19308)

  A run-time error is signaled when SKIP(0) is used as an output control format item in a PUT EDIT statement.

  This problem has been corrected in the DEC PL/I Run Time Library.

- (5-CFS.19740)

  A PUT SKIP before a line written to a PL/I PRINT file is ignored when that line is followed by a PUT PAGE.

  This problem has been corrected in the DEC PL/I Run Time Library.

- (11-CFS.11603)

  A PLI update or install will fail if there is a customer command, LIB included in the DCLTABLES.

  This problem has been corrected the DEC PL/I installation procedure.

- (12-CFS.16104)

Temporary stack space created for built in functions POSINT and INT are not cleaned up properly, possibly leading to ACCVIO or VASFULL.

This problem has been corrected in the DEC PL/I compiler.

- (13-CFS.19495)

  The PL/I COLUMN(n) output control format item was not correctly positioning the columns relative to the current position.

  This problem has been corrected in the DEC PL/I Run Time Library.

- (16-CFS.19311)

  The DEC PL/I compiler was stopping with a %PLIG-F-BUGCHECK fatal error when the right hand side string in a preprocessor statement equals the string in a PICTURE declaration, and the /DEBUG compile command qualifier was used.

  This problem has been corrected in the DEC PL/I compiler.

- (17-CFS.20493)
  (34-CFS.11727)

  Definition of logical name SYSPRINT ignored by PL/1.

  This problem has been corrected in the DEC PL/I Run Time Library.

- (19-CFS.19310)

  The DEC PL/I compiler was stopping with a %PLIG-F-BUGCHECK fatal error instead of reporting a user error when a TYPE clause in a declaration incorrectly refers to an unknown variable.

  This problem has been corrected in the DEC PL/I compiler.

- (20-CFS.19309)

  Fixed length record format sequential files could not be opened for keyed input using integer keys.

  This problem has been corrected in the DEC PL/I Run Time Library.

- (21-CFS.19301)

  Linking a DEC PL/I program resulted in a series of %LINK-W-MULDEF warnings, one for each of the following symbols

  DPLI$MTH_EXT_MOD_DEC4
  DPLI$MTH_EXT_ADD_DEC4
  DPLI$MTH_EXT_SUB_DEC4
  DPLI$MTH_EXT_MUL_DEC4
  DPLI$MTH_EXT_DIV_DEC4
  DPLI$HND_RAISE_CONDITION
  DPLI$MEM_ASSIGN_VSTR

  This problem has been corrected in the DEC PL/I Run Time Library.

- (22-CFS.19143)

  Declarations for the new SYS$GET_SECURITY system service were missing in the PLI$STARLET.TLB text library.

This problem has been corrected in the PLI$STARLET.TLB text library

- (25-CFS.20969)
  (CFS.20969)

  SIGN builtin function returned incorrect results.

  This problem has been corrected in the DEC PL/I Run Time Library.

- (27-CFS.21334)

  DEC PL/I was not properly handling the ENDPAGE condition for file variables in certain cases.

  The compiler allocates a condition handler control block, which is then managed by the PL/I RTL. The compiler was allocating a fixed stack temp for this variable. If an ON statement contains conditions which are not constant (such as ENDPAGE with a file variable), and if the ON statement is inside a loop context, during the second iteration of the loop the information for the first execution of the ON statement was overwritten. This caused the established condition to be "forgotten" by the RTL.

  This problem has been corrected in the DEC PL/I compiler.

- (28-CFS.21453)

  Declarations for new symbols in $CLIDEF were missing in the PLI$STARLET.TLB text library.

  This problem has been corrected in the PLI$STARLET.TLB text library

- (35-CFS.11730,UVO101501)

  A PL/1 Runtime error caused a program abort. Get file list caused a program to crash with an I/O Error.

  This problem has been corrected in the DEC PL/I Run Time Library.

- (36-CFS.16774,MGO100757)

  The FREE statement freed neither the most recent nor any generation of a controlled variable, eventually exhausting memory.

  This problem has been corrected in the DEC PL/I Run Time Library.

- (38-CFS.9384,VNO100006)

  Deleting a nonexistent keyed record caused program to abort with a PL/I internal RAB condition.

  This problem has been corrected in the DEC PL/I Run Time Library.

- (39-CFS.22126)

  When specified in the declaration statement for a file, the MAXIMUM_RECORD_SIZE ENVIRONMENT option was ignored and DEC PL/I opened the file with incorrect attributes.

  This problem has been corrected in the DEC PL/I Run Time Library.

- (44-CFS.23922)

The built in function INDEX returned incorrect results when used on a string with with null characters, such as BYTE(0).

This problem has been corrected in the DEC PL/I Run Time Library.

- (46-CFS.23591)

  The FLOOR built in function performed poorly for fixed binary numbers.

  The function has been improved to provide better performance in the DEC PL/I Run Time Library.

- (47-CFS.23589)

  The DEC PL/I RTL did not expect to encounter a PLI$_ENDPAGE condition which did not have an associated FCB argument (generated using VAXCONDITION(PLI$_ENDPAGE)). This resulted in an access violation in the RTL.

  This problem has been corrected in the DEC PL/I Run Time Library.

- (49-QAR.01808)

  The DEC PL/I PLI$STARTLET.TLB did not define $TPUDEF and $NSADEF include modules correctly.

  This problem has been corrected in the DEC PL/I PLI$STARTLET.TLB text library.

- (51-QAR.1809)

  The BIT( ) builtin produced unexpected result when the argument was neither a string expression nor an arithmetic expression in nonliteral form.

  This problem has been corrected in the DEC PL/I compiler.

- (54-QAR2602)

  The DEC PL/I program using an indexed READ would fail with a "record not found" error.

  This problem has been corrected in the DEC PL/I Run Time Library.

- (71)

  Null bytes throwing off string results in DEC PL/I.

  This problem has been corrected in the DEC PL/I Run Time Library.

- (74-CFS.25858)

  Picture variables could not be output using the F(w) PUT EDIT format item.

  This problem has been corrected in the DEC PL/I Run Time Library.

- (76-CFS.26116)

  The DEC PL/I RTL routine, which is called to establish a condition handler when an ON statement is executed, was not properly handling its data to allow for AST reentrancy. If an ON statement was interrupted and another ON statement was then executed at AST level, an access violation might result.

This problem has been corrected in the DEC PL/I Run Time Library.

- (78-CFS.25914)

  AREA condition not being signaled properly when area was fragmented.

  This problem has been corrected in the DEC PL/I Run Time Library.

- (82-CFS.26326)

  DELETE FILE statement, which is made towards a non-existent record, was causing two signals to be raised.

  This problem has been corrected in the DEC PL/I Run Time Library.

- (84-CFS.26323)

  In a READ FILE statement, the KEY is was not being correctly handled if it was PICTURE data type, causing the READ to fail.

  This problem has been corrected in the DEC PL/I Run Time Library.

- (92-CFS.26832)

  The DEC PL/I compiler was generating incorrect code for select expressions using fixed bin (7) variables when an overflow took place.

  This problem has been corrected in the DEC PL/I compiler.

- (93-CFS.26965)

  A REWRITE FILE statement failed if it was the last record of a sequential file.

  This problem has been corrected in the DEC PL/I Run Time Library.

- (99-CFS.23922)

  The index( ) builtin was returning wrong results when byte(0) occurred in the string.

  This problem has been corrected in the DEC PL/I Run Time Library.

- (134-CFS.28745)

  DEC PL/I SYSPRINT was ignoring the PAGESIZE(n) option of the OPEN statement.

  This problem has been corrected in the DEC PL/I Run Time Library.

- (225-CFS.30137)

  The DEC PL/I compiler was failing to properly record the original picture string and causing a compiler abort when /debug was used.

  This problem has been corrected in the DEC PL/I compiler.

- (243-CFS.30808)

  The DEC PL/I pre-processor was limited to 253 characters string variables

  This problem has been corrected in the DEC PL/I compiler.

10 Overview of New Features in Version 4.0

DEC PL/I Version 4.0 is based on VAX PL/I Version 3.5. DEC PL/I Version 4.0 also includes support for IEEE floating-point data. Version 4.0A also contains the new features of Version 4.0.

10.1 Overview of Language Support

DEC PL/I Version 4.0 is primarily a VAX PL/I Version 3.5 compatibility release. The following sections contain information on support or lack of support for language elements.

10.2 DEBUG Support

DEC PL/I Version 4.0 does not contain full debug support for the following PL/I language data types:

- PL/I unaligned bit-string array variables with run-time sizes and PL/I parameter unaligned bit-string array variables with run-time sizes.

  The following sample program shows an unaligned bit-string array variable:

```
p : procedure options (main);
    call p1 (5);
end;

p1 : procedure (size_ch);
     dcl (size_ch) fixed binary;
     dcl rtb1(size_ch) bit(5);
     dcl rtb2(size_ch,size_ch) bit(5);
end;
```

  In addition, on OpenVMS AXP you cannot correctly examine or deposit to any PL/I unaligned bit-string array variable of unknown size.

- PL/I picture variable data type, as the following program fragment shows:

```
declare p2 picture <picture-specification>;
```

- PL/I varying character variables with run-time sizes, as the following sample program shows:

```
p : procedure options (main);

     call p1 (5);

end;
p1 : procedure (size_chv);
     dcl (size_chv) fixed binary;
     dcl strv1 character (size_chv) varying init('sksks');

     strv1 = 'abcde';
end;
```

In addition, on OpenVMS AXP if a string has run-time bounds the debugger examines and deposits to the variable as if it is a nonvarying string. The work around is to use the debug command "EXAMINE/ASCIW"

- Label array variables.
- Label constants.
- Entry variables and label variables.
- Structure parameter variables of unknown size, specifically PL/I structure parameters declared with star extents or members with unknown size, are not fully supported.
- PL/I external, controlled, defined, and based variables; for example, you cannot examine the externally declared PL/I based variable pli_based_nofp shown in the following sample program:

```
dcl pli_globaldef  fixed binary(31) globaldef;
dcl pli_based_nofp fixed binary(31) based(addr(pli_globaldef));

p : procedure options (main);

end;
```

- Differences in behavior between VAX PL/I and DEC PL/I that involve unreferenced static variables, unreferenced globaldef variables, unreferenced internal static variables, and unreferenced automatic variables.
- Referenced PL/I %REPLACE floating-point constants.
- PL/I REFER structures across all data types.
- Depositing to PL/I bit-string variables using the B format:

```
DBG> DEPOSIT BIT = '111'B
```

The workaround is to deposit without using the B, (i.e. DEPOSIT BIT = '111')

- Referenced PL/I structures declared as globalref. No debugger information is output for the structure members.
- Position attribute for PL/I bit-string variables.

10.3 Differences in VAX PL/I and DEC PL/I Support for Floating-Point Data Types

DEC PL/I Version 4.0 supports five floating-point data types: F, D, G, IEEE S, and IEEE T float. VAX PL/I supports four floating-point data types: F, D, G and H. When you declare a float binary object with a precision greater than 53 (which requires the H floating-point data type), the DEC PL/I compiler supplies a precision of 53 (the maximum value of the G floating-point data type). When you declare a float decimal object with a precision greater than 15 (which requires the H floating-point data type) the DEC PL/I compiler supplies a precision of 15 (the maximum value of the G floating-point data type).

- DEC PL/I may produce truncated results when higher precision values are specified.

  The following PL/I program shows the output from DEC PL/I when precision values exceeding the limits are specified:

```
p : procedure options (main);
    dcl h float binary (113);
    dcl d float decimal (34);
end;
```

  The output received from the DEC PL/I compiler for this program is as follows:

```
      declare d float decimal (34);
........^
%PLIG-W-FLTDPREC, The precision specified for "D" exceeds
        the implementation's limit of FLOAT DECIMAL(15). The
        maximum precision of 15 has been supplied.
        at line number 3 in file tdisk:[tdir]t.pli;1

      declare h float binary (113);
........^
%PLIG-W-FLTBPREC, The precision specified for "H" exceeds
        the implementation's limit of FLOAT BINARY(53). The
        maximum precision of 53 has been supplied.
        at line number 2 in file tdisk:[tdir]t.pli;1

%PLIG-I-SUMMARY, Completed with 0 errors, 2 warnings, and
        0 informational messages.
```

- Before VAX PL/I Version 3.5, certain G and H floating values could improperly be truncated when they were converted to either a fixed-decimal or pictured datatype.

  Even when the PLI command line does not include the /G_FLOAT qualifier and the program does not include floating-point variables large enough to require H float, the VAX PL/I Version 3.5 compiler can generate code that uses H floating data when intermediate calculations require higher precision in order to produce correct results.

  OpenVMS AXP PL/I Version 4.0A does not support a floating-point data type with a precision higher than 53, the maximum value of the G floating-point data type. Therefore, certain conversion operations performed by DEC PL/I may produce results that are truncated in comparison to the results produced by VAX PL/I.

10.4 Support for Other Language Elements

In DEC PL/I Version 4.0A, the format of the ONARGSLIST built-in function is identical to that of VAX PL/I. However, the data structure pointed to by the mechanism array pointer is different.

11 Overview of Changes from VAX PL/I

In addition to the new features outlined above, DEC PL/I Version 4.0 contains the following changes from the VAX PL/I Version 3.5 product.

- Changes to correct VAX PL/I Version 3.5 errors

  A secondary objective of this release is correction of VAX PL/I Version 3.5 errors. A number of problems with Version 3.5 have been corrected. Some of these were documented in the Version 3.5 release notes, while others were discovered after the release of Version 3.5.

- Changes in virtual memory requirements

  The virtual memory requirements for DEC PL/I may be extensive depending to the type of PL/I language constructs that you use. Processing the PUT and GET statements, BEGIN blocks and PROCEDURE blocks with the DEC PL/I compiler tend to require more virtual memory than the same operation would if you were using the VAX PL/I compiler.

  Heavy usage of these constructs in your programs may exhaust your user limits for virtual memory. The typical compiler errors that you receive are:

```
%PLIG-F-TEXT, Compiler abort - virtual memory limits exceeded.
%SYSTEM-F-ABORT, abort
%LIB-F-INSVIRMEM, insufficient virtual memory
```

  You can recover by increasing your user page-file quota (PGFLQUOTA). See the *Kednos PL/I for OpenVMS Alpha Installation Guide* or the OpenVMS System Manager's Manual for more information on increasing your user page-file quota.

- Changes in behavior for calls to bound procedures

  With VAX PL/I, nested procedures can be called by other routines using the PL/I language's entry variable. However, in order to successfully reach these bound-procedure routines and ensure proper execution of uplevel-referenced variables within the bound procedures, the user must keep the parent invocation of the bound procedure active.

  This assures that the stack references to uplevel variables within the bound procedure to the parent's stack, which contained the uplevel-referenced variables, are still valid.

  With DEC PL/I the same situation still exists; however, you may experience different behavior between VAX PL/I and DEC PL/I if you do not keep the parent invocation active. This is due to the fact that bound procedures are not created in the same way by the two compilers.

  DEC PL/I creates the bound procedure values on the stack of the parent of the bound procedure. The user program may produce an access violation or may exhibit other undefined behavior when the parent invocation of the bound procedure is not kept active. Because the stack of the parent has been destroyed, the bound procedure code found on the parent stack may also have been destroyed or corrupted, and the user may not be able to reach the bound procedure.

With VAX PL/I you may receive the same undefined behavior. Cases may exist in which you are able to reach the bound procedure; however, you may still receive errors in results from the uplevel-referenced variables within the bound procedure. This is because the bound procedure code on the parents stack may have been corrupted.

The following example shows behavior where the bound procedure entry6a cannot be reached by DEC PL/I, because the parent procedure, entry6, has exited and its stack is free to be recovered and used by other procedures. Therefore, the bound procedure code has been destroyed.

Note that this programs also fails with VAX PL/I. However, it fails after having reached the bound procedure entry6a and while trying to access the uplevel-referenced variables from the parent, entry6.

Example of Bound Procedure Failure Behavior If Parent Bound Procedure is Inactive

```
program: proc options(main);

  program: proc options(main);
  dcl (ent2) entry variable;
  dcl fnc entry returns(entry) variable;
  dcl evar char (25) var init(' ');

    fnc=entry6;
    ent2=fnc();
    call destroy_stack();
    call ent2;

    put skip list('Evar is =>',evar);

    entry6: proc returns (entry);
    return(entry6a);

      entry6a: proc;
        evar=evar||'entry6a*';
      return;

      end entry6a;

    end entry6;

    destroy_stack: proc;

    /* Declare enough space to destroy previous
     * stack values before this call.
     */

    dcl temp_space char(1000);

      temp_space = 'hello';

    end;
end program;
```

- Changes in behavior for overlapping static storage initialization

  In VAX PL/I and DEC PL/I, overlapping initialization of static storage is not supported. When VAX PL/I finds an overlapping static initialization, the compiler uses the value of the last found initialization as the value for the static variable.

In DEC PL/I you cannot depend on the value of the last found initialization being assigned to the static variable. This behavior may cause a change in behavior for PL/I external variable initializations.

If a PL/I external variable is declared with the attributes EXTERNAL STATIC INITIAL, all blocks that declare the variable MUST be initialized with the same value.

Because VAX PL/I allowed overlapping static storage initialization, you could specify different initial values for the SAME external variable within containing blocks. The last initial value encountered by the VAX PL/I compiler was the value of the external variable.

In DEC PL/I this rule is more strictly enforced: if the user specifies an initial value in the declaration of a PL/I external variable, the same initial value declaration must be specified at each occurrence of the declaration of the external variable in all blocks that declare the external variable. Failure to do so can cause unpredictable results. The following examples illustrate correct and incorrect declarations:

Correct Declaration of an External Variable Initialization

```
program: proc options(main);
dcl test fixed external initial(5);
  p:proc;
  dcl test external initial(5);
  /* The value of this variable should be 5. */
    put skip list ('The value of test is =>',test);
  end;
end;
```

Incorrect Declaration of an External Variable Initialization

```
program: proc options(main);
/* The initialization MUST be the same
 * for all declarations of the external
 * variable.
 */
dcl test fixed external initial(5);
dcl test1 fixed external initial(5);
dcl test2 fixed external;
  p:proc;
  /* The initialization MUST be the same
   * for all declarations of the external
   * variable.
   */
  dcl test external initial(6);
  dcl test1 external;
  dcl test2 external initial(6);
  /* The value of these variables is unpredictable.
   */
```

```
        put skip list ('The value of test is =>',test);
        put skip list ('The value of test1 is =>',test1);
        put skip list ('The value of test2 is =>',test2);
    end;
end;
```

Note: As no guarantee exists as to which occurrence of the external variable the DEC PL/I compiler processes first, each occurrence of the variable must contain the same initial value.

- Changes in behavior for the SUBSTR built-in function

  In PL/I, the SUBSTR function is defined only when the position attribute (the integer expression that indicates the position of the first bit or character in the substring) is greater than or equal to 1.

  If the position attribute is less than 1 (that is, the SUBSTR BIF is undefined), you may observe different results with DEC PL/I than with VAX PL/I.

- Changes in underflow detection behavior for DEC PL/I

  The VAX PL/I compiler outputs underflow detection code, whereas the DEC PL/I compiler does not.

- Changes for null statements and label behavior in DEC PL/I

  VAX PL/I does not compare the multiple labels of many succeeding combinations of labels and null statements so that they have the same address. However, DEC PL/I optimizes these label-null statement sequences and compares the multiple labels so that they result in the same address. The following example illustrates this difference:

```
X67:      PROC options(main);

A:; /* Null statement. */
B:; /* Null statement. */
C:; /* Null statement. */

  IF A=C THEN
    PUT LIST('NULL STATEMENT LABELS COMPARE EQUAL FOR DEC PL/I');
  ELSE
    PUT LIST('NULL STATEMENT LABELS COMPARE NOT EQUAL FOR VAX PL/I');

D:
  IF C=D THEN
    PUT LIST('ERROR THESE LABELS SHOULD NEVER COMPARE EQUAL');

END;
```

- In DEC PL/I you can continue when a GOTO statement with the OTHERWISE option is used to go to an undefined label array element within a BEGIN-END block. VAX PL/I does not support this function. The following example works with DEC PL/I but not VAX PL/I:

```
program: procedure options(main);

dcl i fixed binary(31,0);

begin;
```

```
  i = 2;
  goto part(i) otherwise;
  put skip list('At continue !!');
end;
part(1):
end program;
```

- Run-time exception handling is more consistent when processing formats of GET EDITs and PUT EDITs in DEC PL/I than it is in VAX PL/I. If DEC PL/I encounters an exception when processing a format such as PLI$_INVFMTPARM, a signal is raised. If the exception is handled, then processing continues with the next format item regardless of the nature of the signal and the format item that was being processed when the exception was detected. This is not always the case with VAX PL/I.

  This holds true for conversion errors as well. With VAX PL/I, some conversions with GET EDIT would raise the CONVERSION condition while others would raise the ERROR condition (with ONCODE()=PLI$_CNVERR). DEC PL/I uniformly raises CONVERSION when data cannot be converted. The following example demonstrates this difference.

```
cnverr: proc options (main);
%include $plidef;
declare
   srcstring character(10) initial('4 4445'),
   temp fixed bin(31);
on error begin;
  if oncode() = pli$_cnverr then
    put skip list('VAX PL/I detects CNVERR (not restartable)');
  if oncode() = pli$_oncnvpos then
    put skip list('DEC PL/I detects ONCNVPOS (restartable)');
  goto done;
end;
get string (srcstring) edit (temp) (b3(6));
done:
end cnverr;
```

- Nonlocal returns work properly under DEC PL/I. A nonlocal return is a RETURN statement that is lexically nested between any number of BEGIN/END pairs; for example:

```
nested_proc : proc returns (fixed);
    begin;
        begin;
            begin;
                begin;
                    begin;
                        begin;
                            return (5);
                        end;
                        put skip list ('shouldnt be here 1');
                    end;
                    put skip list ('shouldnt be here 2');
                end;
                put skip list ('shouldnt be here 3');
            end;
            put skip list ('shouldnt be here 4');
    end;
    put skip list ('shouldnt be here 5');
end;
nested_proc2 : proc returns (float);
    begin;
        begin;
            begin;
                begin;
                    begin;
                        begin;
                            return (5.1);
                        end;
                        put skip list ('shouldnt be here 1');
                    end;
                    put skip list ('shouldnt be here 2');
                end;
                put skip list ('shouldnt be here 3');
            end;
            put skip list ('shouldnt be here 4');
    end;
    put skip list ('shouldnt be here 5');
end;

program: proc options(main);

dcl result1 fixed;
dcl result2 float;

  result1 = nested_proc();
  result2 = nested_proc2();

  put skip list (result1);
  put skip list (result2);

end;
```

Two special considerations should be observed for nonlocal returns from OPTIONS(MAIN) procedures. First, an ON unit declared in the scope of one of the BEGIN/END pairs for FINISH is given a chance to execute. Second, an ON unit declared in the scope of one of the BEGIN/END pairs for VAXCONDITION(SS$_UNWIND) is NOT be given a chance to execute with the above construct.

- When a main program (for example, pname : procedure(options(main))) terminates abnormally due to an unhandled exception, DEC PL/I closes all open files before turning control over to the OpenVMS last-chance condition handler (the utility that prints the error and traceback).

This means that I/O that was being held in a buffer is allowed to reach its destination before the error dump appears on the screen. This is not the case with VAX PL/I, in which I/O held in a buffer is not allowed to reach its destination before the error dump appears on the screen.

- Differences in exception handling between OpenVMS AXP and OpenVMS VAX

  Since machine instructions on Alpha AXP(tm) differ from those of the VAX, any exception handler written to handle VAX hardware exceptions should be examined to ensure that it handles Alpha AXP hardware exceptions similarly. For example, consider a case in which an exception handler has been written to handle an access violation. You may expect different behavior upon normal completion of the handler. In this case the handler should always perform a nonlocal GOTO to exit the handler so that program execution continues in a predictable way.

  Note also that exceptions on the Alpha AXP hardware are imprecise and therefore are not always restartable. Digital recommends using a non-local goto to achieve consistent behavior across both PL/I platforms.

- Fixed-decimal precision differences between DEC PL/I and VAX PL/I

  The precision specified for a PL/I fixed-decimal data type must be in the range of 1 to 31 for DEC PL/I. VAX PL/I allows a fixed-point decimal variable to be declared with a precision of zero and also allows built-in functions to specify a fixed-decimal precision of zero. DEC PL/I does not allow zero to be used in either of these situations and issues an error "FIXDPRECZERO" when the precision specified for a fixed decimal is zero.

- Differences in behavior between OpenVMS VAX and OpenVMS AXP architectures regarding PL/I overflow conditions

  In general, any PL/I operation that overflows on OpenVMS VAX also overflows on OpenVMS AXP. Since the Alpha AXP hardware does not include support for packed decimal instructions that correspond to the PL/I FIXED DECIMAL datatype, data items of this type are handled on OpenVMS AXP by run-time calls, to either the DEC PL/I Run-Time Library routines or system Object Time System (OTS) routines. These emulation routines perform many operations to compute the result of a FIXED DECIMAL operation, which in most cases can be done with a single VAX instruction. Many emulation operations generate an overflow.

  Therefore, DEC PL/I guarantees at least one overflow on OpenVMS AXP for every overflow on OpenVMS VAX per PL/I statement. DEC PL/I cannot guarantee that the resulting behavior or value produced by a PL/I statement that produces an overflow condition is the same value or behavior as it is on VAX PL/I.

The following example illustrates a difference in overflow detection between VAX PL/I and DEC PL/I:

```
 23    1        fixb30 = fixd21;

0125  mulp     #10,PLI$B_PAC_2_POWER_30,#31,-60(fp),#31,-96(fp)
0132  ashp     #-21,#31,-96(fp),#0,#31,-112(fp)
013C  cvtpl    #31,-112(fp),r4
0141  movl     r4,-80(fp)
```

The difference occurs when a PL/I fixed decimal data item with a precision of 31 and a scale factor of 21 [FIXED DECIMAL(31,21)] is converted to a PL/I fixed binary data item with a precision 31 and a scale factor of 30 [FIXED BINARY(31,30)]. On OpenVMS VAX this overflow situation results in two overflow conditions being raised. On OpenVMS AXP this situation results in one overflow condition being raised.

Note that all VAX PL/I cases of overflow are detected on OpenVMS AXP in the program. In the example, DEC PL/I detects one overflow for the two overflows reported by VAX PL/I. This difference is due to a difference in the instruction set between and AXP.

The example illustrates two occurrences of this situation. In each case the fixed decimal item is converted to fixed binary item by a series of three steps, as follows:

1 Multiplies the fixed decimal(31,21) item by the decimal representation of 2**30.

2 Shifts the fixed decimal (31,51) created by step 1 to the right by 21.

3 Converts the fixed decimal (31,30) created by step 2 to a fixed binary item (31,30).

The OpenVMS VAX macro instructions output by VAX PL/I to perform this conversion are as follows:

```
23    1        fixb30 = fixd21;

0125  mulp     #10,PLI$B_PAC_2_POWER_30,#31,-60(fp),#31,-96(fp)
0132  ashp     #-21,#31,-96(fp),#0,#31,-112(fp)
013C  cvtpl    #31,-112(fp),r4
0141  movl     r4,-80(fp)
```

In example, during the execution of the OpenVMS VAX *mulp* instruction, and during the OpenVMS VAX *cvtpl* instruction, overflows occur. The OpenVMS AXP instruction set does not contain decimal instructions. Therefore, OpenVMS VAX emulates decimal instructions by means of a series of OpenVMS AXP instructions and OTS calls. During the instructions to emulate the OpenVMS VAX *mulp* instruction, an overflow is correctly detected. During the instructions to convert packed-decimal data to integer data, an overflow is not detected.

Note that after a fixed-overflow condition has been raised, the value resulting from an operation that causes this condition is undefined. In this case the value from the result of the multiplication that caused the overflow is undefined. When it is used in an expression, no guarantee exists that an overflow will be raised again. This happens when the result of the overflow is shifted to the right and then converted from decimal to integer. In this case it is reasonable to expect a difference in the number of overflows detected from one PL/I statement.

Due to the difference between OpenVMS VAX and OpenVMS AXP instructions, this situation cannot be prevented.

In general, on a per-statement basis DEC PL/I detects overflow, but the number of overflows it detects per statement is not guaranteed to be the same on OpenVMS VAX and OpenVMS AXP. The following complete example illustrates the difference:

```
program: procedure options(main);

dcl fixb30  fixed bin(31,30);
dcl fixd18  fixed decimal(31,18);
dcl fixd21  fixed decimal(31,21);
dcl fixd22  fixed decimal(31,22);
dcl fixd24  fixed decimal(31,24);

on fixedoverflow begin;
  put skip list('fixed overflow occurred');
end;

fixd18 = 18.36;
fixd22 = 22.40;

fixd21 = fixd18+fixd22;
fixb30 = fixd21;

fixd18 = 18.42;
fixd24 = 24.58;

fixd21 = fixd24+fixd18;
fixb30 = fixd21;

end;
```

11.1 Corrected Compiler Errors

Corrected compiler errors are as follows:

- VAX PL/I Version 3.5 produced an incorrect result when both the SELECT and WHEN clauses of a SELECT statement contained a substring expression. Local memory for the first substring may have been overwritten by the second substring. This error has been corrected in DEC PL/I Version 4.0.

  The code that produced an incorrect result (the value 1 instead of the correct value of 2) is as follows:

```
p:procedure options(main);

dcl txt char(3);
dcl n_chars fixed binary(31);
dcl option fixed binary(31);

txt = 'NUM';
n_chars = 2;
SELECT (SUBSTR (txt, 1, n_chars));
WHEN (SUBSTR ('LOG', 1, n_chars)) option = 1;
WHEN (SUBSTR ('NUMBER', 1, n_chars)) option = 2;
  OTHERWISE option = 0;
       END;
put skip list ('Option is => ',option);
       END;
```

- VAX PL/I bugchecks when trying to initialize arrays of areas with one PL/I assignment. DEC PL/I has corrected this problem. The following example shows the VAX PL/I behavior:

```
p: procedure options (main);
   dcl array(10) area;
   array = empty();
   end;
```

To avoid this error, initialize arrays of areas by means of a DO loop when using VAX PL/I.

11.2 Corrected Documentation Errors

The errors in this section have been corrected in the current version of the documentation (DEC PL/I Version 4.0).

The VAX PL/I Version 3.5 user manual said on page 2-13 that all Error and Warning error messages are counted toward the error limit. VAX PL/I and DEC PL/I do not count Warning messages, only Error messages, toward the error limit.

The *PL/I Reference Manual for VAX VMS and RISC ULTRIX* contained an error on the description of the LTRIM and RTRIM built-in functions. The definition in that manual for LTRIM(s,[b]) states that LTRIM removes blanks from the left of string s; or if b is supplied, removes string b from the left of string s. The correct definition of LTRIM(s,[b]) is that LTRIM removes white space from the left of string s; or if b is supplied, removes string b from the left of string s. The definition in the manual of RTRIM(s,[b]) states that RTRIM removes blanks from the right of string s; or if b is supplied, removes string b from the right of string s. The correct definition of RTRIM(s,[b]) is that RTRIM removes white space from the right of string s; or if b is supplied, removes string b from the right of string s. White space characters are defined as blank, tab, newline, carriage return, vertical tab, and formfeed.

12 Known Problems and Restrictions With Version 4.1

This section describes known problems with DEC PL/I Version 4.1, and known ways to avoid the problems.

- Problem: Array bounds violations are not detected at run-time even with /CHECK enabled.

  Workaround: Check that code does not exceed boundaries at compile time.

- Problem: Block I/O's of greater than 32K fail. Blocks I/O's of 64k are supported on the VAX version

  Workaround: Check that block I/O's do not exceed the 32k limitation.

13 Known Problems and Restrictions With Version 4.0

This section describes known problems with DEC PL/I Version 4.0. Because DEC PL/I Version 4.0 is based on VAX PL/I Version 3.5, Digital recommends that you read the VAX PL/I Version 3.5 release notes before submitting an SPR.

13.1 Compiler Known Errors and Methods of Avoidance

Known compiler errors and suggestions for avoiding them are as follows:

- In VAX PL/I and DEC PL/I, you can use the OTHERWISE option on a GOTO statement that transfers control to a label-reference that is a subscripted label with a variable subscript. An error is reported if you use the OTHERWISE option on a GOTO statement for a label-reference that is a subscripted label *without* a variable subscript. The latter construct is not recommended, because it causes the compilers to bugcheck. An example of such an inappropriate use of the OTHERWISE option follows:

```
program: procedure options(main);
                goto part(2) otherwise;
        part(2):
                end program;
```

  To avoid this error, use a variable subscript or remove the OTHERWISE option.

- When you use the GOTO statement with the OTHERWISE option, specify label array elements after the GOTO statment. If you use potential label references for a specific GOTO statement prior to the GOTO statement the compiler bugchecks. An example of a program with such a construct follows:

```
program: procedure options(main);
       dcl i fixed binary(31,0);
part(1):
         i = 2;
         goto part(i) otherwise;
end program;
```

To avoid this error, place the label references for a specific OTHERWISE option after the GOTO statement.

- CHARACTER(*) Stack Usage

  Temporary stack space allocated for return values from routines returning CHARACTER(*) is not deallocated until the calling routine exits. The restriction is related to determining the lifetime of the temporary value. A sample code segment displaying this restriction follows:

```
DO I = 1 TO 1000;
   CHARACTER_STRING = ROUTINE_RETURNING_CHAR_STAR();
END;
```

  In this situation, the temporary storage for the return value is not deallocated until the calling routine exits, which results in considerable stack usage. A means of avoiding this restriction is to enclose the use of the routine in a BEGIN block. For example:

```
DO I = 1 TO 1000;
   BEGIN;
   CHARACTER_STRING = ROUTINE_RETURNING_CHAR_STAR();
   END;
END;
```

  Although this method is slightly more CPU-intensive because of the extra CALL/RETURN sequence, it causes the stack top to be correctly reset between each iteration of the loop.

- Silent underflow during compilation

  The compiler silently replaces an underflowing floating-point constant with 0.0 during compilations.

- Descriptors for picture variables

  Picture variables are always passed (incorrectly) by class NRO descriptors regardless of the format of the picture variable. This problem may be addressed in a future release of DEC PL/I.

- The compiler can now handle records up to 32K, which is the RMS maximum. After a file is opened, a buffer equal to the longest record length is allocated. This buffer is then used when the record is read.

  This change is highly dependent on obtaining the longest record length value for a file from RMS (using the xab$w_lrl field of fhc_xab block) when the file is opened. However, rare cases exist in which RMS fails to provide a correct value in that field. If this occurs, the compiler issues the error message %PLIG-F-READERR, which is followed by the operating-system message -RMS-F-USZ, invalid user buffer size.

One alternative is to edit the source file and write out a new version, causing RMS to update the lrl value in the file header.

The compiler issues the error message LOCNEED when the SIZE function references a based variable declared without an associated pointer. For example:

```
TEST: PROC OPTIONS (MAIN);
DCL 1 RECORD BASED,
    3 ITEM1 CHAR(10),
    3 ITEM2 CHAR(20);

PUT EDIT (SIZE(RECORD)) (F(5)); /*  %PLI-E-LOCNEED  */
END;
```

- The compiler issues an incorrect error message when initializing an array through a pointer using an asterisk (*) as an array subscript. The following example illustrates the problem:

```
test: procedure options (main);

dcl 1 tmp1 based,
    2 tmp2 fixed,
    2 tmp (20) bit (17 refer (tmp2) ),
    2 tmp3 fixed;
dcl (mptr,mptr2) pointer;
dcl tmp4 (4) bit (10) based;

   allocate tmp1 set (mptr);
   allocate tmp4 set (mptr2);

   mptr->tmp2 = 17;

   mptr->tmp3 = 15;

   mptr->tmp1.tmp(*) = '10000000001000010'b; /* does not work */
   put skip list (mptr->tmp1);

   mptr2->tmp4(*) = '1000000000'b; /* does not work */
   put skip list (mptr2->tmp4);

end;
```

The error message received is:

```
%PLIG-E-INVSTAREXT, An asterisk is not a valid subscript or
             argument.
```

The alternative is to initialize the array by means of a DO loop.

- VAX PL/I and DEC PL/I compilers bugcheck when trying to pass an asterisk (*) as an argument using the REFERENCE or VALUE built-in functions (BIFs). The problem is caused by a function added to Version 3.5 which allows you to use asterisk subscripts when referencing an entire array.

  The compilers report the following error:

```
%PLIG-E-INVSTAREXT, An asterisk is not a valid subscript or
             argument.
```

Also if the asterisk is passed directly as an argument without using the BIFs, the compilers issue an incorrect error message:

```
%PLIG-E-TOOFEWARG, "Entity" has been referenced with too few
          arguments.
```

If you pass the asterisk using the DESCRIPTOR built-in function, the compilers again issue the wrong error message, as follows:

```
%PLIG-E-DESCRIBIF, Invalid use of the DESCRIPTOR built-in
          function.
```

The compilers should issue the following message in both cases:

```
%PLIG-E-INVSTAREXT, An asterisk is not a valid subscript or
          argument.
```

The following example is a sample of the code to which the compilers respond incorrectly:

```
p: procedure;

dcl e entry (fixed);

call e (reference(*));

end;
```

- The DEC PL/I and VAX PL/I compilers bugcheck if you use an asterisk as a label array subscript, if the variable you use as a subscript for a label array in a GOTO statement is not declared, and if a corresponding label array includes an asterisk. The following PL/I program illustrates the problem:

```
bug: procedure;

goto case1(x);
case1(1):;
case1(*):;
end;
```

Instead, declare the variable explicitly.

- The DEC PL/I compiler sometimes generates an incomplete informational message for PL/I structure variables that are self-referencing. The compiler generates the message when you use the /DEBUG/OPTIMIZE qualifiers. The following PL/I program illustrates the problem.

```
program: PROCEDURE;

DCL A_CURDB EXTERNAL POINTER;

DCL 1 DBCMN BASED ( A_CURDB ),
  2 DBTMAX FIXED BIN (15) ;

DCL LEAF_PTR PTR;

DCL 1 LEAF BASED ( LEAF_PTR ),
          2 LEAF_COUNT FIXED BIN (15),
          2 ENTRY ( LEAF.LEAF_COUNT ),
              3 KEY FIXED BIN (15);

END PROGRAM;
```

The message that you receive does not contain the name of the noninitialized fetch. In this case it is "LEAF.LEAF_COUNT." The message that you receive is:

```
END PROGRAM;
    ^
%PLIG-I-UNINIT, variable  is fetched, not initialized
```

13.2 Run-Time Library Known Errors and Methods of Avoidance

These release notes for DEC PL/I for OpenVMS AXP Version 4.0 include release notes for the OpenVMS AXP Run-Time Library (RTL), as the DEC PL/I for OpenVMS AXP Version 4.0 kit that you receive contains a special version of the Run-Time Library (in the DPLIVMS040 file). In future releases, OpenVMS AXP will supply the RTL, as it has in the past, and RTL release notes will be included with the OpenVMS AXP release notes.

Information in this section is listed by operating-system version:

OpenVMS AXP Version 1.5

- Environment Option FILE_ID_TO

  This option is supported only for temporary files. In addition, the option does not work if the default directory specification is a rooted directory. Digital is unlikely to change this behavior, because this option's rate of use is low. If this restriction constitutes a serious problem for you, please submit an SPR.

- PL/I FINISH condition and DEBUG

  In VAX PL/I, the EXIT command to the debugger raises the FINISH condition and issues a message that the debugger is finishing. DEC PL/I does not raise the FINISH condition due to architectural differences between OpenVMS VAX and OpenVMS AXP.

13.3 Known Errors in the PL/I for OpenVMS Systems Documentation

The known errors in the PL/I for OpenVMS OpenVMS documentation are as follows:

- *PL/I for OpenVMS Systems Reference Manual*

  The title page (the manual accompanying DEC PL/I for OpenVMS AXP Version 4.0) states that the version of the OpenVMS AXP operating system for DEC PL/I is OpenVMS AXP Version 1.5 or higher. This is incorrect. The correct information is OpenVMS AXP Version 1.5.

  Information in Section D.2, Differences Between VAX PL/I and DEC PL/I, is more current in Section 11 of these release notes.

- *PL/I for OpenVMS Systems User Guide*

Figures 10-6 and 11-1 are true for the OpenVMS VAX operating system but not for the OpenVMS AXP operating system, as the manual states. Please see the *OpenVMS Calling Standard* manual (order number AA-PV69A) for the equivalent OpenVMS AXP information. The Calling Standard manual is a component of the Programmer's Kit in the OpenVMS AXP documentation set.

Example 10-8 is true for OpenVMS VAX only, as the manual states. The OpenVMS AXP equivalent is as follows:

Example 1 Displaying Arguments Passed to a Condition Handler (AXP)

```
%INCLUDE $CHFDEF;     ❶
DECLARE X FIXED;
   CHF$ARGPTR = ONARGSLIST();     ❷
/* Output number of signal arguments */
   PUT SKIP LIST('Signal Arg Count',CHF$SIG_ARGS);     ❸
/* Output condition name argument and rest of signal arguments */
   PUT SKIP LIST('Condition name', CHF$SIG_NAME);     ❹
   PUT SKIP LIST(DIM(CHF$SIG_ARG,1),
                 'additional arguments:');     ❺
   DO X = 1 TO DIM(CHF$SIG_ARG,1);
      PUT SKIP LIST(CHF$SIG_ARG(X));
   END;
/* Output RO and R1 */
   PUT SKIP(2) LIST('Low Order 32 bits RO:',CHF$IL_MCH_SAVRO_LOW);     ❻
   PUT SKIP(2) LIST('High Order 32 bits RO:',CHF$IL_MCH_SAVRO_HIGH);
   PUT SKIP(2) LIST('Low Order 32 bits R1:',CHF$IL_MCH_SAVR1_LOW);
   PUT SKIP(2) LIST('High Order 32 bits R1:',CHF$IL_MCH_SAVR1_HIGH);

END;
```

The following notes are keyed to Example 1:

❶ The procedure includes the module $CHFDEF from the default system library.

❷ The ONARGSLIST built-in function assigns a value to the pointer CHF$ARGPTR, declared in $CHFDEF.

❸ Using the CHF$SIG_ARGS field in the signal array, the procedure prints the number of arguments in the signal array.

❹ It displays the contents of the first argument, that is, the condition value.

❺ Because the number of arguments is variable, the procedure uses the DIM built-in function to determine the number of elements in the array CHF$SIG_ARG (this array always contains three fewer members than arguments in the array, because the condition name, PC, and PSL arguments are always present).

❻ After displaying the signal arguments, the procedure displays the contents of R0 and R1 from the mechanism array.

For more detailed information on the argument lists passed to a condition handler and for descriptions of the values in the signal array and mechanism array, see the OpenVMS System Services Reference Manual. Note that the PL/I run-time system signals conditions using the conventions for specifying signal arguments. Specifically, it passes arguments following the requirements described for the SYS$PUTMSG procedure. This procedure is described in the Introduction to the VMS Run-Time Library.

- On pages 2-7 and 2-11, in Section 2.32, the user manual incorrectly states that the /DIAGNOSTICS qualifier to the PLI command is for the OpenVMS VAX operating system only. This qualifier operates on both OpenVMS VAX and OpenVMS AXP.